August 1996

OS-9 International

Vol. 5, Issue 2

startup	3
OS9Lib - Unix C Routines for OS-9	4
OS-9 Network Connectivity via IPX	10
Stack Quo Vadis?	16
sh for OS-9 Version 2.0	18
OS-9 Consulting Offerings	29
Letters to the Editor	30
The EFFO 1996 AGM	32
_getsys();	34

EFFO OS9

European Forum For OS-9

8606 Greifensee, Switzerland
Fax +41 1 940 38 90
email os9int@effo.ch

sFr. 12.00
ISSN: 1019-6714

startup

During the last months, OS-9 International's and EFFO's mail addresses have been used increasingly for support requests. Most of the requests were related to articles in OS-9 International or to EFFO PD software but some requests were even related to more general aspects of OS-9 development systems and system integration.

Up to now, our policy was as follows: whenever time permitted, requests have been answered as rapidly and exhaustively as possible. In earlier days, this was nearly always possible but today, EFFO's resources are no longer sufficient to answer all requests in a way they would merit it. In addition to the higher number of requests, the average work load to answer a request has also increased – mostly because today's software problems are more complex than they were in the past. Another explanation may be that 90 percent of our members are professional software engineers and system designers, whereas five years ago 90 percent of our members were private users. To satisfy the curiosity of the latter was probably easier than to give professional support to the former.

The increasing need for information about OS-9 may further be documented by the positive response to the announcement of EFFO's first conference on OS-9. More than 50 active and passive participants plan to attend this meeting. We here at EFFO will do our very best to make the meeting as informative as possible and are looking forward to it.

As another consequence to the increased number of inquiries for support, OS-9 International has launched a further project to improve support for OS-9. In one of the next issues of the journal, a market survey will be given listing all service providers that offer support for OS-9 and related topics. A questionnaire about any offered software support is, therefore, sent together with this issue of OS-9 International to all subscribers. In addition, this questionnaire is sent to all other companies that are active in the field of embedded systems and software consulting. The questionnaire can also be obtained upon request from EFFO and OS-9 International.

We hope that this activity will help to make life easier not only for the programmer but also for the user who, finally, will get more performant and more reliable OS-9 software.

And now to something completely different. We apologise for the delay of this issue of OS-9 International. We will try our very best that this will never happen again, but we are pretty sure that the next issue will also not be in time due to the OS-9 conference.

Carsten Emde

Reto Peter

Werner Stehling

OS9Lib - Unix C Routines for OS-9

Carsten Emde & the EFFO OS9Lib Working Group[‡]

Introduction

Standardised and portable development platforms are becoming more and more important, partly because of the increasing use of the client/server architecture and partly because of the growing costs of software development. The latter has led to the strategy that every single line of software must be written in such a way that once successfully tested it must run on every reasonable platform. The term 'reasonable platform' normally stands for a 32-bit CPU system with ANSI C compiler and a standard C library. Although there is not one generally accepted standard to describe what a 'standard' C library is, many people would probably agree to define it as the functionality required to compile most of the currently used utilities such as GNU system software, GhostScript and PBM+ image processing packages, BSD network tools and X Window programs. With a few exceptions, the OS-9 kernel supports much of the required functionality so that it was possible to provide a 'standard' C library for OS-9 called *os9lib*. An updated version of this library is now available; it is described in the following article.

History

The first version of the *os9lib* was released as part of the TOP package in 1988, a first update followed about one year later. After this second release, only very few errors have been reported, so that TOP did not release a newer version. Unfortunately, the TOP project has been discontinued in 1991; in consequence, some OS-9 programmers have made additions and modifications to the *os9lib* without sending patches to a common maintainer of the *os9lib* software. From 1993 onwards, the *os9lib* is maintained by EFFO. Since then, it was tried to create a common source pool from the various software versions. In addition, the software was upgraded constantly. Today, most of the EFFO ports of Unix software (e.g. GNU-C/C++, *gmake*, *expr*, *grep*, *sed* etc.) have been produced with the help of the *os9lib*.

[‡]*Marc Balmer, Hans-Werner Bippus, Beat Forster, Pius Meier, Conny Niederhauser, Wolfgang Ocker, Stephan Paschedag, Reto Peter, Wolfgang Reinert, Werner Stehling, Kei Thomsen*

Naming Convention

Initially, the *os9lib* contained both Unix and non-Unix functions. The latter provided additional functionality intended to facilitate programming under OS-9, e.g. accessing the module directory in the same way as if it were a disk directory. The current version of the *os9lib* no longer contains non-Unix functions; they are now available in a second library called *os9ext.l* (OS-9 extension library) that is also contained in the EFFO PD.

Requirements

Most of the functionality required to realise Unix C calls under OS-9 is already available under OS-9 and can be provided without modifying any of the system files. Some functions, however, require that system files are maintained more carefully than it is the case in a normal off-the-shelf OS-9 system. A few other functions require additional system files that can be provided easily. Finally, there are some functions that rely on information provided by additional system tools. These system tools are part of the EFFO *os9lib* distribution and are listed in Table 1. They may be redistributed without any restriction. Special thanks go to Wolfgang Ocker who made available the current *sysinfo* package as public domain software and to Ulli Dessauer for bequeathing *logon* and *mmon* to EFFO. The following overview presents all functions of the *os9lib* separated into the above given four categories.

Name	Function	Replaces
setup	Install sysinfo data base	-
getinfo	Display sysinfo data base	-
logon	User authentication	login
mmon	Time-sharing monitor	tsmon
passwd	Set encrypted password	-
who	List current/recent users	-

Table 1: System tools included with *OS9Lib*.

gethostname	may require the HOST environment variable.
gettz	requires the TZ environment variable (time zone).

Table 3: Functions that may require specific environment settings.

The functions that may be used without any modification are listed in Table 2; functions that may require specific environment settings are listed in Table 3.

Functions that require carefully handled or expanded system files are listed in Table 4. They require that the password file */dd/SYS/password* is managed more consistently than exemplified in Microware's OS-9 distribution. Most importantly, a user ID number must not occur more than once – even not across groups, and the user names must also be unique. In addition, it may be necessary to provide the *gecos* entry that contains additional user information.

_bootdrive	execle_chain	geteuid	popen	strchr
_cmpuid	execlp	getgid	ralarm	strcspn
_console	execv	gethostid	rand	strdup
_doprnt	execv_chain	getopt	randint	strerror
_getpw	execve	getopt_long	random	strpbrk
_strccmp	execve_chain	getopt_long_only	regcomp	strrchr
_strnccmp	execvp	getpass	regexec	strspn
alarm	execvp_chain	getppid	regsub	strstr
alloca10	fchmod	gettimeofday	rename	strtok
atime	fcntl	gtime	rmdir	swab
bcmp	fetch	initstate	set_maxerrnum	sync
bcopy	firstkey	ioctl	seteuid	times
bsearch	forder	isadisk	setgid	truncate
bzero	fstat	isapipe	sethostid	ttyname
crypt	fsync	isatty	setitimer	unix_chown
dbminit	ftime	iswhat	setkey	unix_getgid
ddfopen	ftruncate	mkdir	setstate	unix_getuid
delete	get_maxerrnum	nextkey	shell	unix_setgid
dup2	get_sys_errlist	pclose	signal	unix_setuid
encrypt	get_sys_nerr	perror	srand	utime
errnoprint	getargs	perror_long	srandom	utimes
execl	getcwd	perror_none	stat	xpopen
execl_chain	getdtablesize	pipe	store	xshell
execle	getegid			

Table 2: Functions to be used without any modification.

Functions relying on additional files are listed in Table 5. Three additional files may be required to use these functions:

1. The file */dd/SYS/group* that contains the name of every user group in the format

```
<group name>:<password>:<group ID>:<user>[,<user>]
```

for example:

```
root::0:root
bin::1:root,bin,daemon
daemon::2:root,bin,daemon
sys::3:root,bin,adm
```

2. The file */dd/SYS/termtypes* that contains the type of a terminal connected to a given port in the format

```
<port descriptor> <terminal type>
```

for example:

```
term    vt100
t2      qvt211
```

3. The file */dd/SYS/printertypes* that contains the type of a printer connected to a given port in the format

```
<port descriptor> <printer type>
```

chome
chown
ctermid
cuserid
endpwent
fchown
fgetpwent
getpw
getpwent
getpwnam
getpwopt
getpwuid
setpwent

Table 4: Functions that require carefully handled or expanded system files.

endgrent
fgetgrent
getgrent
getgrgid
getgrnam
getgroups
printertype
setgrent
termtype

Table 5: Functions relying on additional files.

for example:

```
p       fx80
t1      hplj
```

Functions that may require or require additional system tools (e.g. *network*, *setup*, *logon*, *mmon*) are listed in Table 6.

`_utmp_stat`	`getutline`	`info_str`
`_utmp_write`	`getwd`	`info_type`
`endutent`	`getwhoent`	`info_unlock`
`endwhoent`	`info_change`	`pututline`
`findmod`	`info_is_locked`	`setutent`
`gethostname`	`info_kill`	`setwhoent`
`getlogin`	`info_lock`	`sockethostname`
`getutent`	`info_num`	`utmpname`
`getutid`	`info_signal`	

Table 6: Functions that may require or require additional system tools.

Compatibility

Maximum attention was drawn to make the *os9lib* functions as compatible as possible not only to Microware system extensions and tools but also to third party software. The function *gethostname* may serve as an example for this compatibility. It consecutively attempts to retrieve the host name from a variety of different sources and only returns an error condition, if none of them was successful:

```
int gethostname(char *host, int size)
{
  char *envhost, *cp, *hp;
  FILE *fp;

  /* 1. try to determine hostname from socket using SS_GNam */
  host[0] = '\0';
  sockethostname(host, size);
  if (host[0] == '\0') {
    /* 2. try sysinfo database */
    if (info_str("hostname", host, size) == NULL) {
      /* 3. try to get hostname from environment */
      if ((envhost = getenv("HOST")) == NULL) {
        /* 4. try to fork program 'host' and to read its output */
        if ((fp = popen("host", "r")) == NULL ||
                  fgets(host, size, fp) == NULL) {
          if (fp != NULL)
            pclose(fp);
          /* 5. try to fork program 'rpchost' and to scan its output */
          if ((fp = popen("rpchost", "r")) == NULL ||
                    fgets(host, size, fp) == NULL) {
            if (fp != NULL)
              pclose(fp);
            /* all else failed, return error condition */
            errno = E_ILLARG;
            return(-1);
          }
        }
        pclose(fp);
        /* did the program produce something like 'Hostname is: nnn' ? */
        if ((cp = index(host, ':')) != NULL) {
          while (*(++cp) == ' ');
          for (hp = host; *cp != '\0'; )
            *(hp++) = *(cp++);
        }
        /* remove trailing carriage return, if available */
        if ((cp = index(host, '\n')) != NULL)
          *cp = '\0';
      } else
        strncpy(host, envhost, size);
    }
  }
  return(0);
}
```

Test Environment

For the first time, a test environment is available. Every Unix function available in the *os9lib* has its own test function called *test_<function>* that tests at least some of their most usual applications. In addition, the two programs *testos9lib* and *testos9ext* are part of the *os9lib* distribution. These programs make consecutive calls of all supplied test functions.

Conclusion

The above described version of the *os9lib* is now available from EFFO as PD #112. As always, there is no warranty for the program to the extent permitted by applicable law and bug support cannot be guaranteed. EFFO will, however, try to expand, adapt and improve the *os9lib* as much as possible. Bug reports and other requests relative to the *os9lib* should be addressed to *bugs@effo.ch*.

Carsten Emde can be reached at <carsten@effo.ch>.

OS-9 Network Connectivity via IPX

Jürgen Pfeifer, Lothar Albrecht

Introduction

Originally, network connections between OS-9 computers were done using Microware's network protocol OS-9/Net. The OS-9/Net Network File Manager (NFM) allowed for accessing the other computer's I/O devices in the same way as local devices. The disadvantage of OS-9/Net, however, is its restriction to OS-9 systems, since OS-9/Net protocol handlers and NFM client and server software are not available for other operating systems.

The standard Unix network file system (NFS), on the other hand, is non-proprietary software that is available for a wide variety of operating systems including OS-9. One limitation is that only mass storage devices can be shared via network and other I/O devices such as serial devices or pipes can not. Another limitation is that NFS is mainly used on Unix derived operating systems using the TCP/IP protocol. In the past, DOS or Windows based systems were difficult to connect via NFS. Even today, in the year after 95, the large majority of existing DOS and Windows network installations is not based on TCP/IP services but rather on the more popular NetWare software. The latter is based on Novell's Internetwork Packet eXchange (IPX) protocol.

In order to connect an OS-9 system to a Windows computer via IPX, appropriate OS-9 software had to be developed. Such a product is available from Dr. Rudolf Keil GmbH, Dossenheim, Germany; it is called NeWLink. This article gives an overview about the various software components and describes the steps required to develop an example application. This example software will allow to use OS-9 terminals under Windows comparable to *telnet* terminals under X Window.

Prerequisites

OS-9 Computer

The OS-9 computer must be equipped with a network controller interface (Ethernet or Arcnet). The system must run OS-9 version 2.4 or higher. Low-level driver software for the network controller must be available; this software is normally part of the system's software distribu-

tion. In addition, the NeWLink package must be installed consisting of the following components:

- File manager *nwm*,
- Network demon *nwmon*,
- Buffer manager *syskbuf*,
- Service Advertising Protocol (SAP) demon *nwsapd*,
- SAP library *nwsap.l*,
- Sequenced Packet Protocol for IPX (SPX) library *nwspx.l*,
- Hardware-independent high-level driver for IPX, OS-9/Net and/or TCP/IP.

Windows Computer

The Windows computer must be equipped with a corresponding network controller interface (Ethernet or Arcnet). The system must run under Windows 3.1, Windows 3.11 or Windows 95. A Windows NT version is under development. In addition, the NeWLink software package must be available consisting of

- IPX/SPX protocol handler,
- SAP library.

Prerequisites for Target Installation

The related NeWLink target licence is required for the OS-9 target; a standard Novell network software is sufficient under Windows.

Example Application

Problem Definition

The aim of this example application is to provide OS-9 login terminals under Windows. For the sake of simplicity, the maximum number of possible terminal connections is restricted to 10 but, in principle, a more sophisticated solution would not have this limitation. Similar to a *telnet* connection, the OS-9 application should not notice any difference to a physical terminal, i.e. every character entered at the PC is immediately forwarded to the application and any output produced by the application is sent to the terminal emulator, where it is displayed.

Solution

The solution of the above task is, again, similar to the way remote terminals are created using the *telnet* service and is shown in Figure 1.

Providing the Login Network Service

The mechanism of dynamically offering or requesting a network service is defined in Novell's Service Advertising Protocol (SAP). This protocol defines that a service provider informs all network clients about the availability of a given service. The related network packets must contain the provider's station name, socket number and name of the service and the service provider's network address. These packets are regularly sent to the network as broadcast messages. In principle, every system connected to the network can dynamically offer or request such services. In the current application, the OS-9 system is the provider of a terminal login service and the PC is the service requester.

Session Initiation

There are two ways for session initiation by the client:

- **Broadcast message** The network software on the PC waits for the next SAP broadcast message from the OS-9 system. In consequence, it may take a certain time until the next SAP packet arrives.
- **Dedicated request** The PC sends a network packet to the OS-9 system requesting the terminal login service. The OS-9 system then returns a dedicated form of the above described SAP packet to the requester.

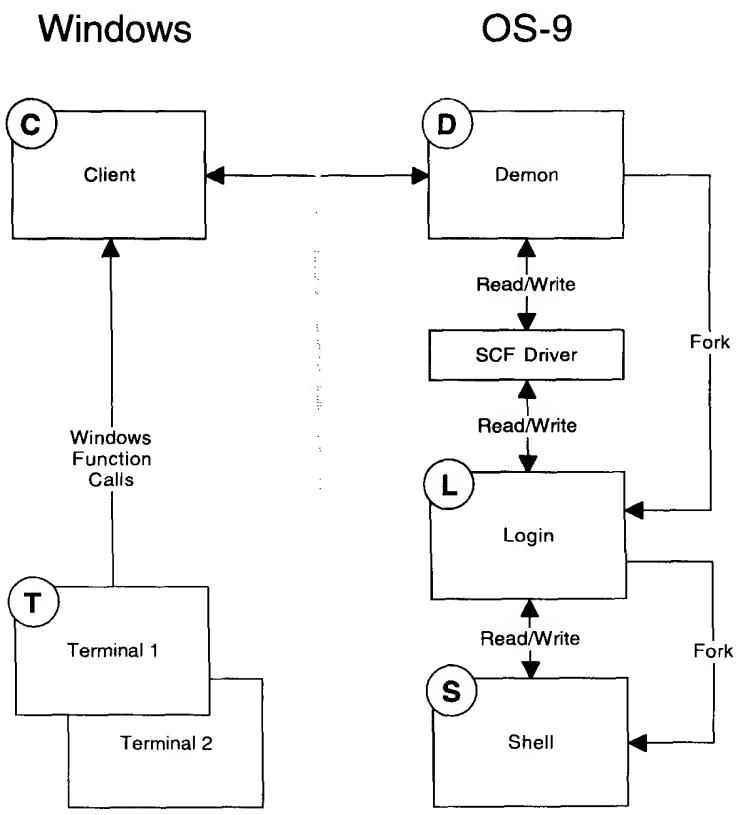

Figure 1: Schematic drawing of the software blocks (see text).

Thus, whenever the PC user wants to login on the OS-9 system, the Windows client (C) in Figure 1 creates a new window on screen (T) and establishes the network connection to the OS-9 demon (D). This can, for example, be realised as shown in the following code section in Visual Basic.

```
'definition of type pc_out
Type pc_out
   cmd     AS Integer
   t_id    AS Integer
   char    AS String * 1
End Type

'global declaration of structure pc_out
Global pc_out  AS pc_out

Sub pbconnect_Click (Value As Integer)

   If termnum >= MAXTERM Then
        MsgBox "Maximum number of terminals reached - request rejected.", 16,
         "Terminal management"
   Else
        pc_out.cmd  = TERMINAL_REQ
        pc_out.t_id = 999  'dummy value, correct value will be defined by demon
        pc_out.char = "a"

        Send_SendData

End Sub
```

Data submission from the client to the demon is realised in a general SPX send subroutine as follows:

```
Sub Send_SendData ()

Dim x As String * 174

   StrucInStr x, pc_out, Len (pc_out)   'copy data from global data structure to
                                        'communication string

   frmmain.spx1.Send = x                'send packet
```

This packet arrives on the OS-9 side, where the OS-9 demon uses the following code in the C language:

```
/* install terminal service */
   if (error=AdvertiseService(servertype, "OS9_TERM_EMU", Socketnum))
        printf("Advertise error %d\n", errno);

/* wait for a connection */
   if(error=listen_con_spx(&con_id, S_name))
        printf("Connect listen error %d\n", error);

/* read from network */
   size=sizeof(struct data_in);
```

```
if(error=read_data_spx(con_id, (ipx_spx_data_buff*) &buff_in.i_s_header,
          &size))
     printf("Read error %d\n", error);
```

The structure element *cmd* is evaluated in a switch command. In the above code example, the value *TERMINAL_REQ* has been sent.

```
switch (Swap16(buff_in.t_header.cmd)) {
     case TERMINAL_REQ:
          buff_in.t_header.t_id = Swap16(termnum);
          buff_in.t_header.cmd  = Swap16(TERMINAL_ID);
```

In a subsequent step, the OS-9 demon creates the *scf* descriptor, the device number of which reflects the requested terminal number, initialises the device and forks the login procedure (L). The latter has its standard input, standard output and standard error paths redirected to the newly created *scf* device. After successful authentication, the login procedure may fork a user shell (S). Only in case everything has completed without error, the terminal creation is acknowledged:

```
          size = sizeof(struct term_header);
          if(error=write_data_spx(con_id, (char*) &buff_in.t_header, &size))
               printf("Write data spx error %d\n", error);
               break;

          /* other cases */
          default: break
     }    /* end switch */
```

The OS-9 demon process not only initiates the network link and starts the login procedure but also creates a data module that contains a ring buffer structure to be accessed by both the demon process and the driver. It also takes care of triggering the transfer of the buffer content to the PC. Finally, the demon sends data to and receives them from the network. The *scf* driver implements all typical functionality of a serial line driver except that no hardware is accessed and no interrupts are used.

For reading and writing, OS-9 uses the same commands as shown above for session initialisation (*read_data_spx* and *write_data_spx*). On the PC, the related commands are *spx.send* and *spx.receive*, respectively. The approach to transfer data from one system to the other is different depending on the direction of the data being sent.

Sending Characters from the PC to the OS-9 System

Data from the PC created by entering characters at the keyboard are transferred one character per packet. Whenever such a character is received by the demon process at the OS-9 system, it is stored in the input buffer; thereafter, a wake-up signal is sent to the *scf* driver. The latter is required in case the OS-9 application is waiting for input using a *read* command.

Sending Data from the OS-9 System to the PC

Data transfer from the OS-9 system to the PC often occurs in larger quantities. In order to keep the network load low and to attain a high data throughput, the data are transferred in packets of up to 512 Bytes using a circular buffer for each terminal window. There are two possibilities to initiate the network transfer of the output buffer contents: first, whenever the buffer becomes full, the driver sends a buffer-full signal to the background process, which in turn sends the content of the buffer via network to the PC. Second, the demon process has installed a cyclic alarm, which may generate a time-out signal. This mechanism takes care of flushing the buffer in regular intervals so that data are sent to the PC, even if the buffer is not yet full.

Session Termination

Two ways exist to terminate a session

- The terminal window on the PC is closed. As a consequence, the PC sends a special code to the OS-9 system causing the current application to be aborted, the device deinitialised and the descriptor removed from memory.
- The user uses the *logout* command. The background process sends a request to the PC to close the terminal window. To close the network session between the two systems, the SPX function *terminate_spx(CON_ID)* can be used.

Conclusion

As exemplified in the above application, a running IPX link between a Windows client and an OS-9 demon can be realised requiring only a few lines of code. In contrast to the described project, more complex applications can certainly be realised having, for example, a state-of-the-art graphical user interface as expected by many users. Up to now, such a graphical user interface required either a local graphics controller on the OS-9 system or the use of network based remote graphics such as X Window. The OS-9 system may not have graphics hardware installed or may be located far away from the operator. The X Window solution may require more system performance and resources than a particular OS-9 system can offer. The NeWLink software is especially helpful, if the integration of a given system requires the use of an already existing Novell network, a Windows-based graphical user interface and a real-time performance that cannot be achieved by a PC.

Jürgen Pfeifer and Lothar Albrecht can be reached via email at <pfeifer@Keil.de> and <alb@Keil.de>, respectively.

Stack Quo Vadis?

Carsten Emde

Problem

In contrast to some other operating systems, OS-9 uses stack space to assign memory to local variables. In consequence, a program that defines

```
main()
{
        char local[4000];
}
```

will require about 4 kByte of stack space and, therefore, not run, if the default stack size of 3 kByte is not increased. The latter can be done using the linker's *-m=<size>* option. The prediction of the required stack is, of course, less simple, if a program is very large and makes many subroutine calls. If the program contains recursive function calls such as in some sort algorithms, it is even more difficult to predict the stack requirements.

Solutions

One solution is to use a dynamic stack as implemented in the current version of EFFO's GNU C compiler by Stephan Paschedag [1]. This feature can be selected by a command line option. The advantage of dynamic stack is that there is no longer any need to care about stack size. As a disadvantage, however, the program may no longer execute at maximum speed. Another more empirical solution is to add the following lines of code to the program to be executed before the program exits

```
#ifdef __STDC__
#define freemem _freemem
#define stacksiz _stacksiz
#endif

main()
{
  func();

  printf("The program '%s' allocated %d Bytes of stack,\n",
    _prgname(), freemem() + stacksiz());
  printf("  a maximum of %d Bytes (%d %%) was used\n",
    stacksiz(), (stacksiz() * 100) / (freemem() + stacksiz()));
}
```

```
func()
{
  char bigstack[2048];
  ;
}
```

and to stress the program with respect to maximum stack allocation. As a rule of thumb, the stack memory requested should be about 2 to 3 kByte larger than the maximum stack depth printed out by the above statement.

Reference

[1] Paschedag S (1995) The GNU C Compilers, Part 3. OS-9 International 4(1):13-20.

Carsten Emde can be reached via email at <carsten@effo.ch>.

sh for OS-9 Version 2.0

Carsten Emde

Introduction

Despite the development of powerful graphic systems such as X Window and MGR, a command line interface between the user and the computer remains an indispensable tool the performance of which continues to contribute to the efficiency of the entire system. This especially applies to development and prototype systems. In the early days of OS-9, only a very rudimentary shell program was available from Microware. Today, an improved shell (*mshell*) can be obtained from Microware but, unfortunately, this one uses a proprietary script language and still does not understand the syntax of shell scripts commonly used in the Unix world. It is, however, not only the desire to execute Unix shell scripts under OS-9; many people also wish to have a shell that incorporates all the useful features from C shell (*csh*), *Korn* shell (*ksh*) and *Bourne-again* shell (*bash*). On the other hand, writing a new shell always is a serious challenge, because the new shell must be fully backwards compatible to the one it is going to replace – nobody wants to use two different shells interchangeably. In the case of OS-9, the challenge was to integrate all features of the original Microware shell into the Bourne-like shell *sh for OS-9*.

Since the last report on *sh for OS-9* [1] that covered version 1.6, edition 32, a number of additions have been made. The current version 2.0, edition 63, is now very close to the ambitious target of creating a fully compatible shell for OS-9. It can be used without problems as the only shell of a system and gives both Unix and OS-9 people the feeling of being 'at home'. In addition, the great majority of Unix Bourne shell scripts, especially those that are implicit part of *makefiles*, can be executed under OS-9 without requiring any adaptation. This greatly facilitates server-based multi-platform software development. Such programming technique that helps to make software independent from a given system architecture is considered, for the time being, an important way of creating future-proof software and, thereby, helping to protect software investment.

This article gives an overview on the current state of *sh for OS-9* with respect to the compatibility to the Microware *shell* and to the various Unix shells. This article, however, cannot give exhaustive information about *sh*'s entire functionality. Such information can be found in the users' manual that is available in electronic and in printed form as part of the public domain distribution of the EFFO PD #107.

Compatibility Issues

The principle goal was to implement important shell language constructs and built-in commands from all currently used Unix and OS-9 shells. In case several Unix shells were incompatible to each other, *bash* was used as reference. In case *shell* language elements such as the pipe symbol '!' or the stack resizer '#' are defined for other purposes in *bash*, a context-sensitive parsing mechanism was implemented. This was, however, not possible in the particular situation of the '>>' output redirection symbol that means appending redirection of standard output path in *bash* and redirection of standard error path in *shell*. To solve this conflict, two operating modes, i.e. the default *Bourne* mode and the optional *shell* mode, have been introduced. Whenever the asterisk is used as comment symbol or a Microware shell option is entered without *set* command, *sh* switches to the *shell* interpretation of the '>>' syntax. The first use of the '#' comment symbol switches *sh* to *Bourne* mode. Thus, a user can force *sh* to start interactive sessions in *shell* mode just by adding a line containing a single asterisk to the *.profile* file.

Compatibility to the Microware Shell

The major version number of *sh* was incremented, because this version for the first time understands Microware's condensed I/O syntax, e.g. *<>>>/nil*, which was *sh*'s major incompatibility to *shell*. The other incompatibilities that still could not be removed are less important and normally not encountered in released shell scripts. A complete list of all changes is provided in the history file *changelog.sh* as part of the delivery of PD #107. There are, however, several compatibility issues that are important enough to be mentioned explicitly:

Missing whitespace

The parsing strategy of the *Bourne* shell script interpreter is, in principle, based on the same mechanism that is used to separate commands and arguments from each other when calling a program, i.e. the end of a word is defined by the first occurrence of a whitespace character. Such whitespace characters are normally space, tab and carriage return/line feed. This becomes very evident, if two or more test conditions are executed consecutively and the results combined using Boolean algebraic operators:

```
if test ( str1 = str2 ) -a ( str1 != str3 -o str1 != str4 )
```

Every parenthesis must be separated from the preceding and the following word by whitespace. In contrast, Microware's shell parser defines the end of a word as the first occurrence of a character that would not be legal in an OS-9 file name. Therefore, a program option or a priority reassigner, for example, can be immediately appended to the program name as in the two commands

```
dir-e
dir^120
```

Unfortunately, *sh* for OS-9 cannot support this linguistic extravagance, since operating systems other than OS-9 may define other characters to be legal file name components. Under Unix, for example, both *dir-e* and *dir^120* are accepted as legal file names.

Handling of unmatched wildcards

sh passes wildcard symbols to the called program, if they do not match at least one file name. Microware's *shell* writes the message "Wildcard match failed for command '<cmd>'" in such a case and does not call the program. The majority of *sh* beta testers, however, have voted in favour of the current approach, that will, therefore, most probably not change in the next future.

Compatibility to Most Unix Shells

Internal execution of grave constructs

Grave constructs (`<command>`) are no longer executed in a separate shell but are evaluated and executed internally. This change was necessary, since a command in the grave construct can, by definition, also be a shell function or an alias that may not be known in a sub-shell environment. As an advantage, this change made grave evaluation considerably faster.

Implementation of new built-in commands

In order to become compatible to most Unix shells, the following commands had to be implemented that were not available in earlier *sh* versions.

- ***builtin <builtin> [<args>]*** Execute the shell built-in command <builtin> passing arguments <args>. Required, if a function has the name of a built-in command, but the built-in command needs to be executed within that function.
- ***dirs*** Display the current stack of remembered directories.
- ***enable [-n] [command]*** This command allows to enable or to disable (*-n*) a shell built-in command so that a disk command with the same name can be executed. If *enable* is entered without arguments, all shell built-in commands are displayed on screen preceded by a '+' or a '-' sign, if the particular command is enabled or disabled, respectively.
- ***popd [+n]*** Removes the entry '*n*' from the directory stack. If *popd* is entered without argument, the top directory is removed from the stack and made the current data directory.
- ***pushd [dir]* or *pushd [+n]*** Adds a directory to the top of the directory stack, if *dir* is a valid directory; otherwise, the directory stack is rotated until the *n*th element of the directory stack is at top. If *pushd* is entered without argument, the top two directories are exchanged. After the rearrangement of the stack, the top directory is always made the current data directory.
- ***suspend [<signal>]*** The shell is suspended until the signal <*signal*> is received. Any activity is suspended except that traps for other signals (installed using the trap <command> <number> command) are executed. If <*signal*> is not specified, any signal will restart the shell.

- ***tty [<devno>]*** Return the name of the path number *<devno>*. If not specified, *<devno>* defaults to 0.
- ***unset [-f] <function>* or *unset <variable>*** Make a previously defined function undefined. If the *-f* option is not specified, *unset* is identical to *unsetenv* except that no error is generated, if an attempt is made to unset a non-existing variable.
- ***unsetenv <variable>*** Make a previously defined environment or shell variable undefined.

Improved error messages

The syntax error message now contains the relative line number and the line in error of the shell script, if *sh* is not interactive. In any case, the column next to the error is clearly marked with the caret symbol (similar to the way *shell* does it). The same applies to errors in the built-in command *test*.

Name completion

Another important addition deals with name completion. In general, name completion means that only the first few characters of a word (command, file name, etc.) need to be typed in. If the completion key (usually *TAB*) is entered, the shell attempts to complete the name based on the word's context. If the entered character sequence is unique, the complete word is inserted into the command line; otherwise, the shell beeps and, if the completion character is hit another time, all matching names are listed on screen.

Five different completion modes have been implemented in *sh*; the first four of them are available in most Unix shells:

- **Command completion** If the first word of a line or of a grave expression is being completed, commands are matched. A command can be an alias, a shell function, a built-in shell command, an executable program in the module directory and, if the *-g* option is specified, a file in one of the execution directories (components of the *PATH* environment variable or current execution directory).
- **Shell variable name completion** If the first character of the word to be completed is a '$' character, shell variables are matched.
- **User name completion** If the first character of the word to be completed is the tilde '~' character, user names are matched.
- **File name completion** If none of the above applies, the entered string is assumed to be the beginning of a file name that is evaluated relative to the current data directory.
- **History line completion** If the first character of a line is the history match symbol '!', a matching history line is searched and, if found, copied into the current command line so that it can be edited. History line completion never reports more than one match, even if there are less recent entries that also match. In contrast to the other completion modes, history line completion is also not limited to words but the entire history line including any white space is inserted. If the history match symbol '!' is followed by a number, the history line at this line number is taken.

In contrast to most Unix shells, *sh*'s name completion even works, if the name to be completed contains a wild-card character or a regular expression.

The not operator

The not operator '!' was implemented. It may appear after *if*, *elif*, *while*, *until*, '||' and '&&'. For example, the command sequence

```
if condition
then
  :  # This is the no-op command
else
  command
fi
```

can now be written as

```
if ! condition
then
  command
fi
```

The syntax is equal to the syntax implemented in *bash* and is intended for better readability of the script code.

The '$@' and the '$*' are handled differently, if in double quotes

If the '$@' argument variable is expanded within double quotes, a separate word is created for every command line argument of the currently executed shell script or function. If the '$*' argument variable is expanded within double quotes, the first character of the Internal Field Separator *IFS* is now used to separate the command line arguments. By default, *IFS* is set to <SPACE><TAB><CR>. In addition, the first character of the *IFS* is now also used to split words after blank expansion.

Shell variable *SH_VERSION*

The unsettable read-only variable *SH_VERSION* containing *sh*'s edition number has been introduced. A shell script, for example, that requires shell edition 52 or higher may contain the following code section

```
required=52
if test -z $SH_VERSION || test $SH_VERSION -lt $required
then
  if test -z $SH_VERSION
  then
    echo Current shell is too old
  else
    echo Current shell is edition $SH_VERSION
  fi
  echo Edition $required required
  echo Please upgrade
  exit 1
fi
```

Returning to the most recent directory

The argument '-' has been added as a legal argument to the *cd* command. The command '*cd -*' is equivalent to '*cd $LWD*', i.e. the most recent directory is made the current directory.

Unix names

In order to allow for running identical shell scripts (e.g. from a server) under Unix and OS-9, the following Unix names are silently transformed:

- A reference to */dev/null* is replaced by */nil*.
- A reference to */tmp* is replaced by the directory specified in the *TMPDIR* environment variable or to */dd/TMP*, if *TMPDIR* is not defined.
- A reference to */bin/<command>* after the '#!' symbol is replaced by the basename, i.e. *<command>*.

Unique Features of *sh for OS-9*

Language Extensions

In addition to some features that make the interactive use of *sh for OS-9* more user-friendly, one general useful language extension has been made to the built-in command *read*, that now accepts input not only from standard input but also from a pipe, which is further explained below. This extension should, of course, better not be used in development environments where the same shell scripts are executed under Unix and under OS-9, unless the same extension is also made to the other shells. The remaining language extensions are specific to OS-9 so that they cannot occur in a script that is intended to be used under both OS-9 and Unix.

The *read* command further enhanced

The built-in command *read* now accepts input from a pipe; this makes it possible to realise a simple string parser as exemplified in the following interactive dialogue:

```
<sh>: echo arg1 arg2 | read a b
<sh>: echo $a
arg1
<sh>: echo $b
arg2
```

Testing whether a given name refers to a memory module

Added *-m* option to the built-in command *test* that returns *true*, if the argument is an existing memory module. Note that this does not imply that a process can link to it.

Script Execution

Starting *sh* scripts from the Microware shell

Introduced *-y* option: if standard input path is not a TTY, standard output and standard error paths are examined. The first of them being a TTY is used as input path for the entire shell session. This feature is required for *sh* scripts that are started from the Microware *shell* and use the *read* command, e.g. for installation purposes.

Automatic Script Execution

More rules for automatic script execution have been added, which allow other programs than *sh* to execute the script. The following rules that are identified by the first character(s) of a script are now available:

First character(s) of script	Executor
#	sh
@	zsh
-	shell
*	shell
~*	cfp
#!<prog>	<prog>

The *cfp* identification characters '~*' make use of the feature that *cfp* defines the tilde '~' as comment character. A *cfp* script that contains the identification characters ~* as the first characters of the file may simply be called by entering its name as if it were a command. The *sh* shell will take care to fork *cfp* and to pass the script name as first argument.

Non-Blocking Read during Command Line Editing

Terminal editing may now be made non-blocking (*-nb*), i.e. *sh*'s standard input device may be used as output by another program even while *sh* interactively waits for the next character to be entered. Default is blocking (the default *SCF* behaviour). This feature is useful, if other programs such as a mail checker may asynchronously write to *sh*'s input port.

Resetting the Terminal's SCF Options

The *kS* termcap capability (default *Ctrl-G*) resets the terminal's *SCF* options to the setting encountered when *sh* was started. If, for example, a program disables the *end-of-record* character and prematurely quits before the *SCF* options are restored, input from command line is no longer possible, unless the reset key is hit.

Kill by Name

The *kill <name>* syntax has been implemented to kill a process by its module name. If the module is running more than once, a message is displayed so that either all processes or none may be killed. This message also contains the affected process IDs and the device names of the first three I/O paths of the affected process IDs so that a particular process can be identified safely.

Memory and CPU Requirements

Despite the named additions and despite an overall functionality that is no longer much inferior to *bash*, the size of *sh for OS-9* is only about 72 kByte, initial memory requirement amounts to about 40 kByte. This was made possible by writing *sh*'s own *cstart* module, to access the operating system by *sh*'s own C bindings and to avoid any buffered and formatted I/O. In addition, algorithms and procedures were carefully optimised to minimise memory and CPU requirements. On the other hand, no CPU-specific optimisations have been made so that *sh for OS-9* runs on any 68-k based OS-9 system including good old MC68000 and any member of the MC683xx family.

Important Enhancements since Vers. 1.6 Ed. 32

Test whether a File is Writable

An interesting finding was that the standard OS-9 library function

```
access("name", S_IWRITE);
```

does an

```
open("name", S_IWRITE);
```

which implicitly updates the date of the file's last modification when the path is closed, irrespective of whether any write action took place or not. This is the reason why

```
test -w name
```

unexpectedly also touched the file. Now, *access* is no longer used so that the last modification date remains unaltered after testing whether a file has write access.

Controlling the History File

Two new modes of the *history* command have been added: when *history read* is entered, the file *~/.sh_history* is read in the same way as when *sh* starts. The current contents of the history buffer are written to *~/.sh_history* when *history save* is entered in the same way as at *logout*. In addition, *history* now writes a usage message, if '-?' or an invalid argument is entered. The *logout* command may now have the argument *nohist*; if this is entered, the contents of the history buffer are not saved. If a user, for example, wants to open an additional *sh* session on another terminal or in another X or MGR window that inherits the history from the existing *sh* session, *history save* can be entered before starting the additional *sh* session. If this history setting is planned to be used by future *sh* sessions after the additional session has been closed, the latter must be quit by explicitly specifying *logout nohist*.

Installing a Data-Carrier-Detect (DCD) Interrupt

In order to make a modem connection more reliable, the *-w* command line option can now be specified. A login *sh* that has been started with this option will attempt to install a signal via the OS-9 C library function *_ss_dcoff*. If this function returns without error, the serial driver will send the specified signal to *sh* in case the DCD line changes from high to low, i.e. the data carrier was lost. The *sh* shell will then kill all child processes recursively and abort with error *E_HANGUP* (220). This makes it possible to equip an OS-9 system with a modem login facility using the standard tool *tsmon* that does not have any modem support.

Miscellaneous

- If the *-l* option is set, *EOF* can now be entered by hitting twice the end-of-file character (normally Escape) at beginning of an input line. *EOF* entered at prompt level causes *sh* to logout; *EOF* entered in *read* command causes the *read* command to abort, to return false and to leave the input variable unchanged.
- *sh* no longer writes nonsense messages such as *nowhere found*, if a shell script contains non-ASCII characters (e.g. German umlauts etc.), since 8-bit characters between 0xc0 and 0xff are now also accepted. If, however, processed by *sh*, the sign bit of the 8-bit character is stripped.
- *sh*'s own *termcap* library is modified in such a way that the *TERMCAP* environment variable is taken from the current set of exported variables. Any modification of the *TERMCAP* environment variable may, therefore, be made available to the current *sh* terminal emulation, if *TERM* is set to the terminal type that is described in the modified *TERMCAP* string.

Frequently Asked Question

Q: If I enter the command

```
break
```

to invoke the ROM level debugger, the message

```
Bad break/continue level
```

appears on screen.

A: Unfortunately, Microware has used the reserved shell and C language word *break* for a command. One solution is to add the line

```
enable -n break
```

to the file *.profile* in the user's home directory and to add the line

```
alias break builtin break
```

to the file */dd/SYS/profile*. Since the local profile is sourced after */dd/SYS/profile* and a non-interactive shell does not evaluate a user's *.profile*, the command *break* will only call the ROM level debugger, if called interactively. As a disadvantage, any script that wants to call the ROM level debugger (which is less usual) must explicitly specify */dd/CMDS/break* and any interactive request of the shell builtin *break* (which also is less usual) must specify *builtin break* instead. Another solution is to provide an *sh* script named, for example, *romdbg* that contains the two lines

```
enable -n break
break
```

so that the command *romdbg* will invoke the ROM level debugger.

Conclusion

OS-9 projects can be developed using one of two very different approaches: self-hosted and cross development. Originally, OS-9 was used predominantly (more than 95%) as a self-hosted development system, i.e. compilation, testing, source file maintenance etc. were all done under OS-9. The same hardware – or a hardware similar to the development system but down-sized – was then used as target system. The most important advantage of such a self-hosted system was the availability of a complete set of dedicated debug tools even on the target systems. Thus, self-hosted systems were often used in hardware-dependent systems that were difficult to simulate on a cross-development platform. In recent years, more cross-development platforms than self-hosted systems have been installed; on average, about 80% of OS-9 development in Europe

is currently still done on self-hosted systems. One argument against using OS-9 as a self-hosted system was the lack of platform-independent tools such as a standard shell, ANSI compilers, a standard *make*, standard source revision systems etc. This argument is no longer true, since all these tools are now available – *sh for OS-9* described in this article is one of them.

Reference

[1] Emde C (1994) An sh-like Shell for OS-9. OS-9 International 3(2):26-33.

Carsten Emde can be reached via email at <carsten@effo.ch>.

OS-9 Consulting Offerings

Hans-Werner Bippus

Rationale

The OS-9 operating system offers a number of standard tools for software development similar to other operating systems. Thus, OS-9 does not impose more difficulties to write platform-independent software than any other system. Unfortunately, a typical OS-9 application is not hardware-independent, it even uses system resources to fullest extent. This includes events and signals to synchronise processes. Shared memory using OS-9 data modules provide common data and trap handlers to reduce memory requirements. In contrast to other operating systems, the required knowledge to handle these specific features is not generally available, but is accumulated in the heads of some few OS-9 system programmers. These coryphaei are either too busy or not allowed to share their expertise with others. The wheel is, therefore, often reinvented at every single company making OS-9 projects more time-consuming and also more expensive than necessary. In the recent past, companies were increasingly willing to purchase external OS-9 know-how, i.e. as a formal training course, as in-house training and consulting or in form of outsourcing. It is rather difficult, however, to gain information about such OS-9 training and consulting offerings.

Solution

EFFO has designed a questionnaire on OS-9 consulting activities that is attached to this issue. If your company can provide software support related to OS-9 or related to tools that are commonly used under OS-9, please complete the questionnaire and return it to EFFO. One of the next issues of OS-9 International will present the returned information in detail. In addition, this information will be used whenever requests for software support are submitted to EFFO being beyond the scope of EFFO's honorary capability.

EFFO kindly invites you to complete the attached questionnaire.

Hans-Werner Bippus can be reached via email at <hwb@effo.ch>.

Letters to the Editor

Remote Commands and Network Time Services for OS-9

OS-9 International 1/95, p. 23

Recently, I received the EFFO *Remote Command* disk and tried to install the *rsh* mechanism to start an *xterm* on our OS-9 system remotely from my PC. Unfortunately without success. The only message I am constantly receiving is "Permission denied". The explanation in the related article in OS-9 International is, sorry to say, not terribly helpful. Could you provide a more detailed explanation of how the *rsh* authentication mechanism works? As there seem to be some differences between the Unix *rsh* mechanism and the OS-9 implementation of the *rsh* mechanism, could you also go into that in more detail? Remark: I am working with OS-9 2.4, ISP 2.0, *rsh/rshd/rshdc* Edition #10 of 29.3.95.

Hermine Arnold, Joanneum Research, Graz, Austria <arnold@pdibm.joanneum.ac.at>.

We apologise for the confusion created in the article on remote commands under OS-9. Here comes another attempt to describe the authentication procedure incorporated in the remote commands. In compliance with general practice, we have not implemented features that will behave different from their Unix versions. As an addition, *rshd*'s debug and trace output has been improved as shown below.

User Authentication

As a general principle, user authentication can be divided into two generally different procedures depending on whether the user wants to become superuser on the server system or not.

1. The user wants to become superuser on the server system.

Only the file *.rhosts* in the superuser's home directory on the server is checked but the file *hosts.equiv* is never checked. The file *.rhosts* must be owned by the superuser, must have at least private read access and must not have write access for anyone else than the superuser. At least one of the lines in the file *.rhosts* must contain the name of the calling host in the first column; otherwise, permission to use one of the remote commands is not granted. If the host name is found, it is checked whether it is a fully qualified host name including the host domain or not. If not, the host domain is queried from the system and appended. If the line with the matching host name does not contain more than a single column, the authentication is successful, if the client user name is the same as the server user name. If it contains more columns, every word is matched against the client user name and the authentication procedure is only successful, if the user name can be found. In other words, the client user name is only allowed to be different from the server user name, if specified explicitly.

2. The user does not want to become superuser on the server system.

In a first step, the file *hosts.equiv* is checked. The lookup procedure is identical to the one described above: If i) the host is listed, ii) the host domain name matches and iii) the client user name is the same as the server user name or the client user name can be found in the list of user names, permission is granted. Otherwise, the user's *.rhosts* file is checked. Again as explained above, the file *.rhosts* must be owned by the specified user, must have at least private read access, must not have public write access and must contain the client host name followed or not by user names as above. If this second authentication step also fails, permission is denied to use any of the remote commands.

Software upgrade

In order to facilitate debugging of the authentication procedure, the respective functions have been rewritten entirely. A complete protocol is now output to */dd/LOG/rshd.log*, if the highest debug level (*-dddddd*) is specified in the command line of *rshd*.

The following example protocol has been generated when the user *carsten* (user ID 200.200) at host *thllin* executed the command *procs -e* on host *thlmak* as user *weo* (user ID 201.201) using the command

```
rsh thlmak -l weo '@procs -e'
```

Contents of the file */dd/LOG/rshd.log* on host *thlmak*:

```
rshd: Starts at Sun Jun  2 14:10:03 1996
rshd: Connect from 192.52.109.60 @ Sun Jun  2 14:10:03 1996
...
rshd: trying to check-in user 'carsten' from client host 'thllin.ceag.ch' as normal user
rshd: user 'carsten' is user 'weo' on the server system
rshd: client host name 'thllin.ceag.ch' transformed to 'thllin.ceag.ch' length 6
rshd: trying to open file '/dd/SYS/hosts.equiv' ... success
rshd: checking client host 'thllin.ceag.ch' in file '/dd/SYS/hosts.equiv'
rshd: domain qualifier found
rshd: client host 'thllin.ceag.ch' does not match host field 'thlpci.ceag.ch'
rshd: domain qualifier found
rshd: client host 'thllin.ceag.ch' exactly matches host field 'thllin.ceag.ch'
rshd: user 'carsten' is different from user 'weo'
rshd: found user 'weo' with home at '/h0/USR/weo' in server's password file
rshd: successfully opened file '/h0/USR/weo/.rhosts'
rshd: got stat info of file '/h0/USR/weo/.rhosts'
rshd: good, uid of file '/h0/USR/weo/.rhosts' (201) is the same as the user id 201
rshd: good, file '/h0/USR/weo/.rhosts' is not public writable
rshd: checking client host 'thllin.ceag.ch' in file '/h0/USR/weo/.rhosts'
rshd: domain qualifier found
rshd: client host 'thllin.ceag.ch' exactly matches host field 'thllin.ceag.ch'
rshd: comparing client user 'carsten' with specified user 'carsten' ... passed
rshd: _validuser() returns 0 to indicate successful validation
rshd: executing 'procs -e'
```

Carsten Emde <carsten@effo.ch>.

The EFFO 1996 AGM

Reto Peter, EFFO Secretary

The EFFO annual general meeting took place on Saturday, 9th March 1996, 14:45 to 19:40 at the Gasthof zur Herberge in Teufenthal (near Aarau). All registered members of EFFO had received a written invitation including the proposed agenda and the proposed budget for 1996.

The president, Werner Stehling, welcomed the members present to the 9th general assembly of EFFO. He reported on the past year starting with expressing his thanks to all active members of EFFO. The number of members has increased considerably to as much as 125. The observed and already mentioned shift from private users towards companies continued, and today only very few of the EFFO members are private users.

The considerable increase in the number of members is primarily based on support activities from Eltec and PEP, who continued to inform their customers about EFFO and to offer a free one-year membership to those customers who purchased an OS-9 development system.

The orders of PD disks have doubled during the last year. An average of four titles were requested per order. The number of PD software packages has grown from 15 to 20. These additional five titles include three of the four packages announced in last year's protocol.

Three issues of EFFO's journal OS-9 International have been published in 1995 at the scheduled dates. This is especially remarkable as still only very few articles are contributed by external writers. One more positive aspect is the larger number of advertisements placed in OS-9 International, although some of them created unexpected work load to integrate them into the layout and to ensure adequate printing quality.

Formal Points of the Agenda

The profit and loss account and the balance for the last year were presented and checked by the auditor and accepted unanimously upon his request.

The elections of the officers were performed rapidly, and all nominees were elected with the maximum of votes possible. The members of the committee for 1996 are:

President	Werner Stehling (as before)
Vice-president	Reto Peter (as before)
Secretary / registrar	Reto Peter (as before)
Treasurer	Stephan Paschedag (as before)
Chief editor	Carsten Emde (as before)
Auditor	Hans-Werner Bippus (as before)

Two problems have been discussed on the financial side. First, there are the fees for banking transactions, especially if foreign currencies are involved. In some cases, up to 25 percent of the money sent to EFFO has been retained by the bank to cover transfer fees. In nearly all of these cases, Swiss Franks have been sent to the German account or vice versa. In order to avoid these unnecessary costs, EFFO asks its customers to exclusively send German currency to the German account and Swiss currency to the Swiss account. EFFO decided to accept credit cards to cope with this problem and also to facilitate the payment of orders from overseas. For the time being, however, Visa is the only credit card that EFFO can handle to receive payment for PD disks and membership fees.

The second problem was the drastic raising of prices for postage, which is also considered in the budget for 1996. As a consequence, the prices for PD orders do not include postage any more; it will be charged separately. The prices for OS-9 International still include handling and postage fees, but they needed to be adapted.

The annual EFFO membership fee remains unchanged, members from Eastern Europe continue to get a free EFFO membership.

Miscellaneous

The two main items planned for 1996 are the preparation and publication of another three issues of OS-9 International and the organisation of the first EFFO OS-9 conference. The issue 3/96 of OS-9 International will contain an article reporting on the outcome of this conference. As an encouraging result of our activities, a considerable number of participants have already registered.

Regarding EFFO's PD software packages, the following enhancements are planned:

- Perl version 4.3, already finished, but not yet released,
- a universal plot package for graphical output in a variety of different formats,
- a largely enhanced version of the OS9Lib,
- the commonly used revision control system RCS.

Please note that these disks may only be ordered after they appear on our list of officially released PD software packages.

EFFO plans to offer its services on the World Wide Web. Some activities are, therefore, started to set up our own home page. More details will be given as soon as the service is available.

Reto Peter can be reached at <reto@effo.ch>.

_getsys();

Reto Peter

Monthly EFFO Meetings

The monthly EFFO meeting takes place each first Friday of a month in the Restaurant Palmhof in Zurich. Its exact address is Universitätsstr. 23, CH–8006 Zurich, phone +41 1 261 69 90. It can easily be reached from the main railway station using tram 10. As usual, the meeting starts at 8 PM, but most participants meet at 7 PM in the Restaurant to have supper together. Everybody interested in OS-9 is kindly invited to join the meeting.

EFFO 's OS-9 Conference

The already announced OS-9 conference organised by EFFO takes place in Emmetten, Switzerland, from September 20 until September 22, 1996. More than 50 participants have registered.

Imprint

OS-9 International

Published by	European Forum For OS-9 (EFFO)
President	Werner Stehling
Vice President	Reto Peter
Director of Finance	Stephan Paschedag
Editor-in-Chief	Carsten Emde
Design	Marc Balmer, Werner Stehling (layout)

Address
European Forum For OS-9
P.O. Box
8606 Greifensee
Switzerland

FAX +41 1 940 38 90
email os9int@effo.ch

Printed directly from disk by Fotoplast, Zurich, Switzerland
ISSN: 1019–6714

Subscriptions

OS-9 International is the official organ of the European Forum For OS-9 (EFFO). The subscription is included with the annual EFFO membership fee. In addition, it is available by separate subscription for non-EFFO members, single issues are also available. All following prices are given in Swiss Francs, shipping included:

	Switzerland	*Europe*	*Overseas*
One year (3 issues)	26.00	37.00	45.00
Single issue	12.00	15.50	18.00

To subscribe to OS-9 International or to order a single issue send a letter, postcard, fax or email to EFFO

Advertisements

OS-9 International is not only an ideal platform for discussing OS-9 related topics, it is also the ideal place to advertise. OS-9 International reaches end-users, system-software developers and, nevertheless, decision-makers.
Please contact EFFO for detailed information on how to place an ad in OS-9 International.